# Apple Assembly Line

$1.80

Volume 7 -- Issue 12 September, 1987




Gary Little's new book is finally here: "Exploring the Apple IIgs", published by Addison-Wesley, 534 pages. If you are seriously trying to conquer the IIgs in assembly language, this book is a must. Instead of just telling us the tool numbers, Gary includes working examples of assembly language code which use the tools. Here is a list of the chapter titles, which I think is the easiest way to describe the contents:

1. Exploring the Apple IIgs
2. Programming the 65816 Microprocessor
3. Using the GS Tools
4. Memory Management
5. Event Management
6. Windows and Graphics
7. Using Pull-Down Menus
8. Using Dialog and Alert Boxes
9. All about Desk Accessories
10. The ProDOS-16 Operating System
11. Sound and Music
12. Using the Text Tool Set

The examples are all written using Apple's APW assembler, and make extensive use of the toolbox macros included with that assembler. A coupon in the back of the book lets you order a complete source code disk directly from Gary Little for all of the programs listed in the book for only $20. You can buy the book from us for $21 plus shipping.

Another new book this month is also from Addison-Wesley, in the official Apple IIgs series, "The Apple IIgs Hardware", 288 pages + 8 foldout schematics. This one lists at $24.95, and you can get it from us for $23. Again, if you are serious about programming the IIgs in assembly language you probably need this book. It contains information about the hardware you won't find in the other books.

## Draw Circles the Fast Way.................Bob Sander-Cederlof

I loved the article "Vector-to-Raster Algorithms" by Dick Pountain in the September 1987 issue of BYTE. It is the kind of article that explains why I have been a loyal subscriber for ten years or so. Pountain carefully explains how the fundamental graphics subroutines are written, and gives pseudo-code examples for each one. And, wonder of wonders, BYTE printed all of the examples! (It has been so frustrating the last three years to find that the examples must be downloaded from BIX or ordered at extra cost!)

I decided to try my hand at coding a fast circle-drawing subroutine for Apple hi-res graphics. Things like that are already included in the //gs QuickDraw ToolBox for Super Hi-Res, but I am not aware of any generally available circle-drawers for ordinary hi-res.

Since I wrote the double lo-res article last month, I decided to first try adding a circle-drawer to that program. It would be somewhat easier, since all of the variables involved could be 8-bit integers. But even before that I decided to write an Applesoft version, using normal hi-res.

```
 1000  REM  DRAW A CIRCLE USING MICHENER'S ALGORITHM
 1010  HGR
 1020  HCOLOR= 3
 1030  INPUT CX,CY,R
 1040 X = 0:Y = R:E = 3 - 2 * R
 1050  REM PLOT 8 POINTS
 1060  HPLOT CX + X,CY + Y: HPLOT CX + X,CY - Y
 1070  HPLOT CX - X,CY + Y: HPLOT CX - X,CY - Y
 1080  HPLOT CX + Y,CY + X: HPLOT CX + Y,CY - X
 1090  HPLOT CX - Y,CY + X: HPLOT CX - Y,CY - X
 1100  REM ADVANCE TO NEXT POINT
 1110  IF E > 0 THEN E = E + 4 * (X - Y) + 10:Y = Y - 1
       : GOTO 1130
 1120 E = E + 4 * X + 6
 1130 X = X + 1: IF X <  = Y THEN 1060
 1140  GOTO 1030
```

This program will draw a circle of radius R and center at CX,CY. Line 1030 lets you enter the center and radius. The algorithm actually computes an X-offset and Y-offset from the center for each point in 1/8th of the circle. Lines 1060-1090 use the offsets to plot eight points, one in each of the half-quadrants. The three expressions for calculating E, in lines 1040, 1110, and 1120 are the secret to the program. Pountain says they were derived by "algebraic re-arrangement" of the basic equation of a circle: X^2 + Y^2 = R^2. Maybe so, I'll just take his word for it.

I found it interesting to play with some of the parameters. I changed line 1030 do assign specific values to CX, CY, and R and then ask for a value "A". Then I changed the expression for "E" in line 1110 by substituting "A" in the place of "4". By changing the value of "A" I could draw anything from a square to a diamond with missing points. Try it!

```
S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners...................$20
ProDOS Upgrade Kit for Version 2.0 DOS owners........................$30
Source Code of S-C Macro 2.0 (DOS or ProDOS)........each, additional $100
S-C DisAssembler (ProDOS only)...without source code $30, with source $50
RAK-Ware DISASM (DOS only).......without source code $30, with source $50
ProVIEW (ProDOS only)................................................$20
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
S-C Word Processor (with complete source code).......................$50
DP18 and DPFP, including complete source and object code.............$50
ES-CAPE (Extended S-C Applesoft Program Editor)..........
                     Including Version 2.0 With Source Code..........$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)....$30
Bag of Tricks 2 (Quality Software)......................($49.95)  $45 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00)  $50 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
                8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)
Beagle Bros. Applesoft Compiler (ProDOS only)...............($74.95)  $65 *
Copy II Plus (Central Point Software).......................($39.95)  $30 *
SoftSwitch (Roger Wagner Publishing)........................($59.95)  $48 *

AAL Quarterly Disks.........................each $15, or any four for $45
 Each disk contains the source code from three issues of AAL:
   Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings, sleeves, labels)..............box of 10 for $6 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100 *
Envelopes for Diskette Mailers............................  6 cents each

Sider 20 Meg Hard Disk, includes controller & software........($695) $550 +
Minuteman 250 Uninterruptible Power Supply....................($359) $320 +
65802 Microprocessor (Western Design Center)............................ $25 *
quikLoader EPROM System (SCRG)................................($179) $170 *
PROmGRAMER (SCRG).........................................($149.50) $140 *

"Exploring the Apple IIgs", Gary B. Little..................($22.95)  $21 *
"Apple IIgs Technical Reference", Fischer...................($19.95)  $19 *
"65816/65802 Assembly Language Programming", Fischer........($21.95)  $20 *
"Programming the 65816", Eyes & Lichty......................($22.95)  $21 *
"Apple //e Reference Manual", Apple Computer................($24.95)  $23 *
"Apple //c Reference Manual", Apple Computer................($24.95)  $23 *
"ProDOS-8 Technical Reference Manual", Apple Computer.......($29.95)  $27 *
"ProDOS-16 Technical Reference Manual", Apple Computer......($29.95)  $27 *
"Apple IIgs Firmware Reference", Apple Computer.............($24.95)  $23 *
"Apple IIgs Hardware Reference", Apple Computer.............($24.95)  $23 *
"ProDOS Inside and Out", Doms & Weishaar....................($16.95)  $16 *
"DOStalk Scrapbook", Weishaar & Kersey......................($14.95)  $14 *
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95)  $18 *
"Beneath Apple DOS", Worth & Lechner........................($19.95)  $18 *
"Inside the Apple //c", Little..............................($19.95)  $18 *
"Inside the Apple //e", Little..............................($19.95)  $18 *
"Understanding the Apple //e", Sather.......................($24.95)  $23 *
"Understanding the Apple II", Sather........................($22.95)  $21 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($18.95)  $18 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95)  $18 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95)  $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95)  $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20 *
"Microcomputer Graphics", Myers.............................($14.95)  $14 *
"Assembly Lines -- the Book", Wagner........................($19.95)  $12 *

            * On these items add $2.00 for the first item and
              $.75 for each additional item for US shipping.
            + Inquire for shipping cost.
              Foreign customers inquire for postage needed.
        Texas residents please add 8% sales tax to all orders.

        *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***


        ***                   (214) 324-2050                     ***
        *** Master Card, VISA, Discover and American Express ***
```

Next I wrote the double lo-res version. The listing which follows can be appended to the listing on pages 7-10 in the August article. After assembly, executing the program named TC will clear the screen and draw four circles. They are drawn fast enough that they appear almost instantaneously.

Lines 3270-3330 define a macro named CIRCLE, which make it easy to call the CIRCLE subroutine. The three parameters are simply CX, CY, and R. SInce this is double lo-res, the legal limits will be small. CX may be 0 to 79, CY may be 0 to 47, and R may be 0 to 23. Any larger R-value would force part of the circle to go off the screen. I did not put in any error checking in this version, so if you do try to go off the edge you will probably clobber something. In my hi-res version I put in code to avoid plotting off the screen.

The code is basically a hand-compilation of the Applesoft version. The comments show the equivalent Applesoft statements.

```
                  3270 *--------------------------------
                  3280        .MA CIRCLE
                  3290        LDA #]3     R
                  3300        LDX #]1     CX
                  3310        LDY #]2     CY
                  3320        JSR CIRCLE
                  3330        .EM
                  3340 *--------------------------------
                  3350 TC
09BE- 20 8A 08    3360        JSR DLR     Turn on Double Lo-Res
09C1- 20 DF 08    3370        JSR CLRTOP  Clear screen
09C4- A9 0F       3380        LDA #15     Set Color = White
09C6- 20 64 F8    3390        JSR SETCOL
                  3400 *--------------------------------
09C9-             3410        >CIRCLE 40,20,19
09C9- A9 13       0000>        LDA #19     R
09CB- A2 28       0000>        LDX #40     CX
09CD- A0 14       0000>        LDY #20     CY
09CF- 20 F8 09    0000>        JSR CIRCLE
09D2-             3420        >CIRCLE 40,27,4
09D2- A9 04       0000>        LDA #4      R
09D4- A2 28       0000>        LDX #40     CX
09D6- A0 1B       0000>        LDY #27     CY
09D8- 20 F8 09    0000>        JSR CIRCLE
09DB-             3430        >CIRCLE 30,15,3
09DB- A9 03       0000>        LDA #3      R
09DD- A2 1E       0000>        LDX #30     CX
09DF- A0 0F       0000>        LDY #15     CY
09E1- 20 F8 09    0000>        JSR CIRCLE
09E4-             3440        >CIRCLE 50,15,3
09E4- A9 03       0000>        LDA #3      R
09E6- A2 32       0000>        LDX #50     CX
09E8- A0 0F       0000>        LDY #15     CY
09EA- 20 F8 09    0000>        JSR CIRCLE
                  3450 *--------------------------------
09ED- AD 00 C0    3460 .2     LDA $C000
09F0- 10 FB       3470        BPL .2
09F2- 8D 10 C0    3480        STA $C010
                  3490 *--------------------------------
09F5- 4C 80 08    3500        JMP TEXT
                  3510 *--------------------------------
                  3520 CIRCLE
09F8- 8E 95 0A    3530        STX CX      get center coordinates
09FB- 8C 96 0A    3540        STY CY
09FE- 8D 8F 0A    3550        STA Y1      Y = R
```

```
                  3560 *--- D = 3 - 2*R ---------------
0A01- 0A          3570        ASL          2*R
0A02- 49 FF       3580        EOR #$FF     -2*R-1
0A04- 69 04       3590        ADC #4       3-2*R
0A06- 8D 94 0A    3600        STA D
                  3610 *--- X = 0 --------------------
0A09- A9 00       3620        LDA #0
0A0B- 8D 8E 0A    3630        STA X1
                  3640 *---equiv. to lines 1050-1090----
0A0E- AE 8E 0A    3650 .1     LDX X1
0A11- AC 8F 0A    3660        LDY Y1
0A14- 20 51 0A    3670        JSR PLOT.FOUR.POINTS
0A17- AE 8F 0A    3680        LDX Y1
0A1A- AC 8E 0A    3690        LDY X1
0A1D- 20 51 0A    3700        JSR PLOT.FOUR.POINTS
                  3710 *-------------------------------
0A20- AD 94 0A    3720        LDA D        IF D<=0 THEN GO TO .2
0A23- 30 13       3730        BMI .2
0A25- F0 11       3740        BEQ .2
0A27- 38          3750        SEC          D = D + 4*(X-Y) + 10
0A28- AD 8E 0A    3760        LDA X1
0A2B- ED 8F 0A    3770        SBC Y1
0A2E- 0A          3780        ASL
0A2F- 0A          3790        ASL
0A30- 18          3800        CLC
0A31- 69 0A       3810        ADC #10
0A33- CE 8F 0A    3820        DEC Y1
0A36- D0 07       3830        BNE .3
                  3840 *-------------------------------
0A38- AD 8E 0A    3850 .2     LDA X1       D = D + 4*X + 6
0A3B- 0A          3860        ASL
0A3C- 0A          3870        ASL
0A3D- 69 06       3880        ADC #6
0A3F- 6D 94 0A    3890 .3     ADC D
0A42- 8D 94 0A    3900        STA D
0A45- EE 8E 0A    3910        INC X1       X = X + 1
0A48- AD 8F 0A    3920        LDA Y1       IF Y <= X THEN GO TO .1
0A4B- CD 8E 0A    3930        CMP X1
0A4E- B0 BE       3940        BCS .1
0A50- 60          3950        RTS
                  3960 *-------------------------------
                  3970 PLOT.FOUR.POINTS
0A51- 8E 92 0A    3980        STX DX
0A54- 8C 93 0A    3990        STY DY
0A57- 18          4000        CLC          PLOT CX+DX,CY+DY
0A58- AD 95 0A    4010        LDA CX
0A5B- 6D 92 0A    4020        ADC DX
0A5E- A8          4030        TAY
0A5F- AD 96 0A    4040        LDA CY
0A62- 6D 93 0A    4050        ADC DY
0A65- 20 A5 08    4060        JSR DPLOT
0A68- 38          4070        SEC          PLOT CX+DX,CY-DY
0A69- AD 96 0A    4080        LDA CY
0A6C- ED 93 0A    4090        SBC DY
0A6F- 20 A5 08    4100        JSR DPLOT
0A72- 38          4110        SEC          PLOT CX-DX,CY+DY
0A73- AD 95 0A    4120        LDA CX
0A76- ED 92 0A    4130        SBC DX
0A79- A8          4140        TAY
0A7A- 18          4150        CLC
0A7B- AD 96 0A    4160        LDA CY
0A7E- 6D 93 0A    4170        ADC DY
0A81- 20 A5 08    4180        JSR DPLOT
0A84- 38          4190        SEC          PLOT CX-DX,CY-DY
0A85- AD 96 0A    4200        LDA CY
0A88- ED 93 0A    4210        SBC DY
0A8B- 4C A5 08    4220        JMP DPLOT
                  4230 *-------------------------------
0A8E-             4240 X1     .BS 1
0A8F-             4250 Y1     .BS 1
0A90-             4260 SX     .BS 1
0A91-             4270 SY     .BS 1
0A92-             4280 DX     .BS 1
0A93-             4290 DY     .BS 1
0A94-             4300 D      .BS 1
0A95-             4310 CX     .BS 1
0A96-             4320 CY     .BS 1
                  4330 *-------------------------------
```

Next I wrote a normal hi-res version. The listing that follows is a stand-alone program, but you could adapt it to work with Applesoft or to incorporate the CIRCLE subroutine in your own program. I used three subroutines in the Applesoft ROM to do the actual plotting, so the Applesoft ROMs have to be switched on for the program to run. Since I ran it from within the DOS 3.3 version of my assembler, I had to put line 1180 at the beginning to turn on the ROM. Line 1430 puts it back to the language card RAM.

The demonstration part of this program starts at line 1170. I used a macro named CIRCLE again, but the definition is slightly different (see lines 1080-1140). Since the X-coordinate can now be from 0 to 279, it will take two bytes to store. Therefore I wrote my macro to first pick up the two bytes of the X-coordinate in X and Y, get the single byte of the Y-coordinate in A, and then call CENTER to store them in some standard variables. Then I pick up the radius in A and call CIRCLE. The demo draws two "faces" and two circles which go off the edge of the screen. The latter two demonstrate the feature I added to detect off-screen points and bypass plotting them.

The Applesoft ROM subroutines I used are HGR, HCOLOR, and HPLOT. HGR turns on the primary hi-res screen and clears it. HCOLOR is inside the HCOLOR= processor, and expects a color value (0...7) in the X-register. HPLOT expects the X-coordinate in the A-register, and the Y-coordinate in the Y- and X-registers. Calling HPLOT is much faster in machine language, because all reference to floating point values is avoided.

I wrote my own caller for HPLOT in lines 2590-2720. It first checks the coordinates to see if they are on the screen. If not, the subroutine returns without trying to plot. If they are on the screen, I save X and Y and call HPLOT. I saved X and Y here so that I could use them again to plot another point with the same X-coordinate.

I also added a tiny bit of error checking to the CIRCLE subroutine. If the radius is larger than 127 line 1560 catches it and rings the bell. I could have handled larger radii, but it would have required more of the arithmetic to be done in 16-bit precision.

The rest of the code in CIRCLE is very similar to the double lo-res version. I had to go to 16-bit precision for the "D" calculations, and that is the main change.

```
                  1000 *SAVE S.HIRES.CIRCLES
                  1010        .OP 65C02
                  1020 *--------------------------------
F3E2-             1030 HGR    .EQ $F3E2
F457-             1040 HPLOT  .EQ $F457
F6F0-             1050 HCOLOR .EQ $F6F0
FBDD-             1060 MON.BELL .EQ $FBDD
                  1070 *--------------------------------
```

```
                  1080        .MA CIRCLE
                  1090        LDA #]2       CY
                  1100        LDX #]1       CX-LOW
                  1110        LDY /]1       CX-HI
                  1120        JSR CENTER
                  1130        LDA #]3       R
                  1140        JSR CIRCLE
                  1150        .EM
                  1160 *--------------------------------
                  1170 TC
0800- AD 81 C0    1180        LDA $C081     SELECT APPLESOFT ROMS
0803- 20 E2 F3    1190        JSR HGR       Turn on HI-RES
0806- A2 03       1200        LDX #3        Set Color = White
0808- 20 F0 F6    1210        JSR HCOLOR
                  1220 *--------------------------------
080B-             1230        >CIRCLE 40,20,19
080B- A9 14       0000>        LDA #20       CY
080D- A2 28       0000>        LDX #40       CX-LOW
080F- A0 00       0000>        LDY /40       CX-HI
0811- 20 AC 08    0000>        JSR CENTER
0814- A9 13       0000>        LDA #19       R
0816- 20 B9 08    0000>        JSR CIRCLE
0819-             1240        >CIRCLE 40,27,4
0819- A9 1B       0000>        LDA #27       CY
081B- A2 28       0000>        LDX #40       CX-LOW
081D- A0 00       0000>        LDY /40       CX-HI
081F- 20 AC 08    0000>        JSR CENTER
0822- A9 04       0000>        LDA #4       R
0824- 20 B9 08    0000>        JSR CIRCLE
0827-             1250        >CIRCLE 30,15,3
0827- A9 0F       0000>        LDA #15       CY
0829- A2 1E       0000>        LDX #30       CX-LOW
082B- A0 00       0000>        LDY /30       CX-HI
082D- 20 AC 08    0000>        JSR CENTER
0830- A9 03       0000>        LDA #3       R
0832- 20 B9 08    0000>        JSR CIRCLE
0835-             1260        >CIRCLE 50,15,3
0835- A9 0F       0000>        LDA #15       CY
0837- A2 32       0000>        LDX #50       CX-LOW
0839- A0 00       0000>        LDY /50       CX-HI
083B- 20 AC 08    0000>        JSR CENTER
083E- A9 03       0000>        LDA #3       R
0840- 20 B9 08    0000>        JSR CIRCLE
                  1270 *--------------------------------
0843-             1280        >CIRCLE 140,40,39
0843- A9 28       0000>        LDA #40       CY
0845- A2 8C       0000>        LDX #140       CX-LOW
0847- A0 00       0000>        LDY /140       CX-HI
0849- 20 AC 08    0000>        JSR CENTER
084C- A9 27       0000>        LDA #39       R
084E- 20 B9 08    0000>        JSR CIRCLE
0851-             1290        >CIRCLE 140,47,8
0851- A9 2F       0000>        LDA #47       CY
0853- A2 8C       0000>        LDX #140       CX-LOW
0855- A0 00       0000>        LDY /140       CX-HI
0857- 20 AC 08    0000>        JSR CENTER
085A- A9 08       0000>        LDA #8       R
085C- 20 B9 08    0000>        JSR CIRCLE
085F-             1300        >CIRCLE 130,35,3
085F- A9 23       0000>        LDA #35       CY
0861- A2 82       0000>        LDX #130       CX-LOW
0863- A0 00       0000>        LDY /130       CX-HI
0865- 20 AC 08    0000>        JSR CENTER
0868- A9 03       0000>        LDA #3       R
086A- 20 B9 08    0000>        JSR CIRCLE
086D-             1310        >CIRCLE 150,35,3
086D- A9 23       0000>        LDA #35       CY
086F- A2 96       0000>        LDX #150       CX-LOW
0871- A0 00       0000>        LDY /150       CX-HI
0873- 20 AC 08    0000>        JSR CENTER
0876- A9 03       0000>        LDA #3       R
0878- 20 B9 08    0000>        JSR CIRCLE
                  1320 *--------------------------------
087B-             1330        >CIRCLE 140,60,80
```

```
0889-                        >CIRCLE 260,60,30
0889- A9 3C     0000>        LDA #60       CY
088B- A2 04     0000>        LDX #260       CX-LOW
088D- A0 01     0000>        LDY /260       CX-HI
088F- 20 AC 08  0000>        JSR CENTER
0892- A9 1E     0000>        LDA #30       R
0894- 20 B9 08  0000>        JSR CIRCLE
                      *--------------------------------
0897- AD 00 C0        .2     LDA $C000
089A- 10 FB                  BPL .2
089C- 8D 10 C0               STA $C010
                      *--------------------------------
089F- 8D 51 C0               STA $C051     TEXT
08A2- 8D 5F C0               STA $C05F     SINGLE
08A5- 8D 54 C0               STA $C054
08A8- 8D 80 C0               STA $C080     BACK TO S-C MACRO IN RAM
08AB- 60                     RTS
                      *--------------------------------
                      CENTER
08AC- 8D BC 09               STA CY
08AF- 8E BA 09               STX CX
08B2- 8C BB 09               STY CX+1
08B5- 60                     RTS
                      *--------------------------------
08B6- 4C DD FB        BEEP   JMP MON.BELL
                      CIRCLE
08B9- 8D B5 09               STA Y1        Y1 = R
08BC- 0A                     ASL           2*R
08BD- B0 F7                  BCS BEEP      ...RADIUS TOO LARGE
08BF- 8D B8 09               STA D         D = 3 - 2*R
08C2- 38                     SEC
08C3- A9 03                  LDA #3
08C5- ED B8 09               SBC D
08C8- 8D B8 09               STA D
08CB- A9 00                  LDA #0
08CD- 8D B4 09               STA X1        X1 = 0
08D0- E9 00                  SBC #0
08D2- 8D B9 09               STA D+1
                      *--------------------------------
08D5- AE B4 09        .1     LDX X1
08D8- AC B5 09               LDY Y1
08DB- 20 58 09               JSR PLOT.FOUR.POINTS
08DE- AE B5 09               LDX Y1
08E1- AC B4 09               LDY X1
08E4- 20 58 09               JSR PLOT.FOUR.POINTS
                      *--------------------------------
08E7- AD B9 09               LDA D+1       IF D <= 0 THEN GO TO .2
08EA- 30 25                  BMI .2
08EC- 0D B8 09               ORA D
08EF- F0 20                  BEQ .2
                      *--------------------------------
08F1- 38                     SEC
08F2- AD B5 09               LDA Y1
08F5- ED B4 09               SBC X1
08F8- 20 47 09               JSR TIMES.FOUR
08FB- 38                     SEC           10 - 4*(Y-X)
08FC- A9 0A                  LDA #10
08FE- ED BD 09               SBC TEMP
0901- 8D BD 09               STA TEMP
0904- A9 00                  LDA #0
0906- ED BE 09               SBC TEMP+1
0909- 8D BE 09               STA TEMP+1
090C- CE B5 09               DEC Y1
090F- D0 14                  BNE .3        ...ALWAYS
                      *--------------------------------
0911- AD B4 09        .2     LDA X1        6 + 4*X
0914- 20 47 09               JSR TIMES.FOUR
0917- 18                     CLC
0918- AD BD 09               LDA TEMP
091B- 69 06                  ADC #6
091D- 8D BD 09               STA TEMP
0920- AD BE 09               LDA TEMP+1
0923- 69 00                  ADC #0
                      *--------------------------------
0925- 8D BE 09        .3     STA TEMP+1
0928- 18                     CLC           D = D + TEMP
0929- AD BD 09               LDA TEMP
092C- 6D B8 09               ADC D
092F- 8D B8 09               STA D
```

```
0932- AD BE 09             LDA TEMP+1
0935- 6D B9 09             ADC D+1
0938- 8D B9 09             STA D+1
093B- EE B4 09             INC X1
093E- AD B5 09             LDA Y1
0941- CD B4 09             CMP X1
0944- B0 8F                BCS .1
0946- 60              .99  RTS
                      *--------------------------------
                      TIMES.FOUR
0947- A0 00                LDY #0
0949- 8C BE 09             STY TEMP+1
094C- 0A                   ASL
094D- 2E BE 09             ROL TEMP+1
0950- 0A                   ASL
0951- 2E BE 09             ROL TEMP+1
0954- 8D BD 09             STA TEMP
0957- 60                   RTS
                      *--------------------------------
                      PLOT.FOUR.POINTS
0958- 8E B6 09             STX DX
095B- 8C B7 09             STY DY
095E- 18                   CLC          CX+DX,CY+DY
095F- AD BA 09             LDA CX
0962- 6D B6 09             ADC DX
0965- AA                   TAX
0966- AD BB 09             LDA CX+1
0969- 69 00                ADC #0
096B- A8                   TAY
096C- AD BC 09             LDA CY
096F- 6D B7 09             ADC DY
0972- 20 9E 09             JSR CALL.HPLOT
0975- 38                   SEC          CX+DX,CY-DY
0976- AD BC 09             LDA CY
0979- ED B7 09             SBC DY
097C- 20 9E 09             JSR CALL.HPLOT
097F- 38                   SEC          CX-DX,CY+DY
0980- AD BA 09             LDA CX
0983- ED B6 09             SBC DX
0986- AA                   TAX
0987- AD BB 09             LDA CX+1
098A- E9 00                SBC #0
098C- A8                   TAY
098D- 18                   CLC
098E- AD BC 09             LDA CY
0991- 6D B7 09             ADC DY
0994- 20 9E 09             JSR CALL.HPLOT
0997- 38                   SEC          CX-DX,CY-DY
0998- AD BC 09             LDA CY
099B- ED B7 09             SBC DY
                      ***  JMP CALL.HPLOT
                      *--------------------------------
                      CALL.HPLOT
099E- C9 C0                CMP #192
09A0- B0 11                BCS .2       OFF THE SCREEN
09A2- C0 01                CPY /280
09A4- 90 06                BCC .1       ON THE SCREEN
09A6- D0 0B                BNE .2       OFF THE SCREEN
09A8- E0 18                CPX #280
09AA- B0 07                BCS .2       OFF THE SCREEN
09AC- DA              .1   PHX
09AD- 5A                   PHY
09AE- 20 57 F4             JSR HPLOT
09B1- 7A                   PLY
09B2- FA                   PLX
09B3- 60              .2   RTS
                      *--------------------------------
09B4-                 X1   .BS 1
09B5-                 Y1   .BS 1
09B6-                 DX   .BS 1
09B7-                 DY   .BS 1
09B8-                 D    .BS 2
09BA-                 CX   .BS 2
09BC-                 CY   .BS 1
09BD-                 TEMP .BS 2
                      *--------------------------------
```

Some IIgs Demos....................................Bob Urschel

I thought some of my fellow AAL readers might like these two programs I wrote for my IIgs. Much of the information I needed to write them came from the excellent book by Gary Bond, "Inside the Apple IIgs", from Sybex.

The graphics demo program will display all of the 4096 colors that are available on the Apple IIgs, 256 colors at a time. Pressing any key except ESCAPE will display the next 256 colors. There are 16 pages altogether. Pressing ESCAPE will exit the program.

The little loop in lines 1480-1580 clears the Super Hi-Res picture buffer. Another way which you might think would be faster is to use the MVN instruction with an overlapping move:

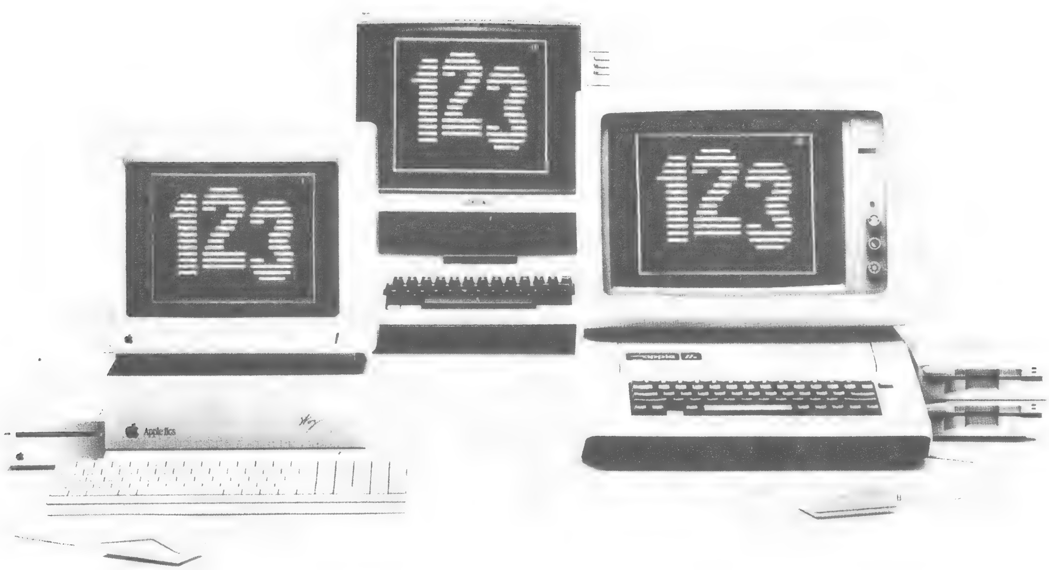
```
       STZ $2000
       LDA ##$7FFE
       LDX ##$2000
       LDY ##$2001
       MVN $E10000,$E10000
```

However, this takes longer. The MVN takes 7 cycles per byte moved, and we are moving 8191 bytes. The loop I used takes 13 cycles for each of 4096 byte-pairs. That is a savings of about 4096 cycles, but that is not all of the story. MVN is both loading and storing bytes in bank $E1, which is a slow-ram bank. Two cycles out off every seven must run at 1 megahertz instead of 2.8 megahertz. In my loop only two cycles out of every 13 are in slow RAM. I'll let you figure out exactly what this means in microseconds.

Notice above when I wrote the MVN instruction I used 24-bit values to specify the banks. A more intuitive MVN $E1,$E1 will not work. This is because Bob S-C figured the natural thing to do would be to use the actual label names for the areas being moved, and these would be 24-bit addresses. For example:

```
       SHR.PIC .EQ $E12000
               STZ >SHR.PIC
               LDA ##8190
               LDX ##SHR.PIC
               LDY ##SHR.PIC+1
               MVN SHR.PIC,SHR.PIC
```

The sound demo program makes use of the IIgs toolbox. The sound wave is generated by the Applesoft program listed below. I BSAVEd the sine wave data after running the Applesoft program. From the S-C Assembler I loaded and assembled the sound demo program, then BLOADed the sine wave data, and then typed "MGO S" to play the sine wave through the Ensoniq chip. Pressing any key stops the tone.




# Now Apple speaks IBM. Three times faster than IBM.

**Introducing PC Transporter.™ The Apple® II expansion board that lets you run MS®-DOS programs.**

Now your Apple II can run over 10,000 programs you could never use before. Like Lotus® 1-2-3.® MultiMate.® dBASE III PLUS.® Even Flight Simulator.®

With PC Transporter, MS-DOS programs run on your Apple II like they do on IBM® PC's or compatibles. With one important difference. PC Transporter runs most of those programs *three times faster* than an IBM PC/XT.®

Plus, to speed through number-crunching tasks, you can use our optional 8087-2 math co-processor chip. It plugs into a socket on the PC Transporter.

**Less expensive than an IBM clone.**

Sure, a stripped-down IBM clone costs about the same as the PC Transporter. But the peripherals it takes to get the clone up and running make the clone cost about three times what our American-made card costs.

You don't have to buy new hardware to use PC Transporter.

**Works with the hardware you already own.**

With PC Transporter, MS-DOS programs see your Apple hardware as IBM hardware. You can use the same hardware you have now.

With IBM software, your Apple hardware works just like IBM hardware. Including your drives, monitors, printers, printer cards, clock cards and serial clocks.

You can use your IIe® or IIGS™ keyboard with IBM software. Or use our optional IBM-style keyboard (required for the II Plus).

You can use your Apple mouse. Or an IBM compatible serial mouse.

**Plenty of power.**

PC Transporter gives you as much as 640K of user RAM and 128K of system RAM in the IBM mode.

PC Transporter also is an Apple expansion card, adding up to 768K of extra RAM in the Apple mode. The Apple expansion alone is a $300 value.

**Easy to install.**

You can install PC Transporter in about 15 minutes, even if you've never added an expansion board. You don't need special tools. Simply plug it into an Apple expansion slot (1 through 7 except 3), connect a few cables and a disk drive, and go!

```
                   *-------------------------------
                   *      Super hires 256 color demo
                   *      8-6-87 ru
                   *
                              .op 65816
                   *
9D00-              scb.buff   .eq      $9d00   Scanline Control Buffer start
2000-              pix.buff   .eq      $2000   Start of pixel buffer
9E00-              pal.buff   .eq      $9e00   Palette data
C000-              kbd        .eq      $c000
C010-              kbd.strb   .eq      $c010
C029-              shr.sw     .eq      $c029  super hires switch
C034-              border     .eq      $C034  border color stored here
                   *-------------------------------
000800- 18         s          clc              native mode
000801- FB                    xce
000802- A9 C1                 lda #$c1         turn on super hires
000804- 8D 29 C0              sta shr.sw
000807- AD 34 C0              lda border
00080A- 48                    pha              save border color
00080B- 29 F0                 and #$f0         set border to black
00080D- 8D 34 C0              sta border
000810- A9 E1                 lda #$e1         super hires in bank $e1
000812- 48                    pha
000813- AB                    plb
000814- C2 30                 rep #$30         16 bit regs
                   * clear shr buffer
000816- A2 FE 7F              ldx ##$7ffe      clear 32k
000819- 9E 00 20   .1         stz pix.buff,x
00081C- CA                    dex
00081D- CA                    dex
00081E- 10 F9                 bpl .1
                   *-------------------------------
                   *  set up SCB's
                   *  12 lines per SCB
                   *
000820- A9 00 00   setscb     lda ##0          start with palette #0
000823- A2 00 00              ldx ##0
000826- A0 06 00   .2         ldy ##6          repeat for 12 display lines (a twofer)
000829- 9D 00 9D   .1         sta scb.buff,x     index into SCB buffer
00082C- E8                    inx              double inx since acc = 2 bytes
00082D- E8                    inx
00082E- 88                    dey              done yet?
00082F- D0 F8                 bne .1           no, else..
000831- 18                    clc
000832- 69 01 01              adc ##$0101      set up SCB for next palette
000835- C9 10 10              cmp ##$1010      16 palettes yet?
000838- D0 EC                 bne .2           no
                   *-------------------------------
                   *          set up pixel data
                   *
00083A- A2 00 00   setpix     ldx ##0
00083D- A9 00 00   .3         lda ##0
000840- A0 05 00   .2         ldy ##5          320 mode/16 per line = 20 bytes
000843- 9D 00 20   .1         sta pix.buff,x
000846- E8                    inx
000847- E8                    inx
000848- 88                    dey              done 20 pixels yet?
000849- D0 F8                 bne .1           no, else..
00084B- 18                    clc
00084C- 69 11 11              adc ##$1111      get next palette number
00084F- C9 10 11              cmp ##$1110      16 palette colors yet?
000852- D0 EC                 bne .2           no, else..
000854- E0 00 78              cpx ##$7800      end of display buffer yet?
000857- D0 E4                 bne .3           no, else..
```

```
                           2260 *--------------------------------
                           2280 *      set up 256 colors in palettes
                           2300 *
000859- A9 00 00           2320 setcol lda ##0
00085C- A2 00 00           2340 .2     ldx ##0      first palette
00085F- 9D 00 9E           2360 .1     sta pal.buff,x    save color
000862- 18                 2380        clc
000863- 69 01 00           2400        adc ##1      inc color value
000866- E8                 2420        inx
000867- E8                 2440        inx
000868- E0 00 02           2460        cpx ##$200   fill 512 byte palette table
00086B- D0 F2              2480        bne .1
00086D- 48                 2500        pha          save acc for next 256 colors
00086E- E2 20              2520        sep #$20     8 bit acc
000870- AD 00 C0           2540 .3     lda kbd
000873- 10 FB              2560        bpl .3
000875- 8D 10 C0           2580        sta kbd.strb      clear keyboard strobe
000878- C9 9B              2600        cmp #$9b     esc?
00087A- F0 05              2620        beq end
00087C- C2 20              2640        rep #$20
00087E- 68                 2660        pla          get last color+1
00087F- 80 DB              2680        bra .2       display next 256 colors
                           2700 *--------------------------------
                           2720 end
000881- E2 30              2740        sep #$30     back to 8 bit regs
000883- 68                 2760        pla          clean up stack
000884- 68                 2780        pla
000885- A9 00              2800        lda #0
000887- 48                 2820        pha
000888- AB                 2840        plb          back to bank $00
000889- 38                 2860        sec          emulation mode
00088A- FB                 2880        xce
00088B- A9 01              2900        lda #$01
00088D- 8D 29 C0           2920        sta shr.sw   turn off super hires
000890- 68                 2940        pla          get back border color
000891- 8D 34 C0           2960        sta border
000894- 60                 2980        rts
```

```
                  1000 *-------------------------------
                  1020 *      Sound Demo
                  1040 *      8-1-87 ru
                  1060 *
                       1080          .OP 65816
E10000-                1100 Toolbox     .eq        $e10000
                       1120 *
000800- 18             1140 S        clc           Native Mode
000801- FB             1160          xce
000802- C2 30          1180          rep #$30      16 bit registers
000804- F4 00 30       1200          pea $3000     GCB'S (Used by Toolbox: 256 bytes long)
000807- A2 08 02       1220          ldx ##$0208   SoundStartup Tool
00080A- 22 00 00 E1 1240             jsl Toolbox
00080E- F4 01 00       1260          pea $0001     push the generator number and $01
000811- F4 00 00       1280          pea $0000     Push location of param table:
000814- F4 00 03       1300          pea $0300     4 bytes - z,b,h,l
000817- A2 08 0E       1320          ldx ##$0e08   StartSound tool call
00081A- 22 00 00 E1 1340             jsl Toolbox
00081E- E2 20          1360          sep #$20      set a reg to 8 bits
000820- AD 00 C0       1380 .1       lda $c000     Wait for keypress
000823- 10 FB          1400          bpl .1
000825- AD 10 C0       1420          lda $c010     Reset keyboard strobe
000828- C2 20          1440          rep #$20      set a reg to 16 bits
00082A- F4 7F FF       1460          pea $ff7f     push the mask to stop all generators
00082D- A2 08 0F       1480          ldx ##$0f08   StopSound tool
000830- 22 00 00 E1 1500             jsl Toolbox
000834- A2 08 03       1520          ldx ##$0308   SoundShutdown tool
000837- 22 00 00 E1 1540             jsl Toolbox
00083B- 38             1560          sec
00083C- FB             1580          xce           back to emulation mode
00083D- E2 30          1600          sep #$30
00083F- 60             1620          rts
                       1640 *-------------------------------
                       1660 *        parameter table
                       1680          .or $300
000300- 00 20 00 00 1700             .hs 00.20.00.00   Wave Table addr in main ram: l,h,b,z
000304- 04 00          1720          .hs 04.00         wave table is four pages long
000306- 00 0A          1740          .hs 00.0A         frequency: l,h
000308- 00 00          1760          .hs 00.00         Wave Table addr in dedicated ram: l,h
00030A- 02             1780          .hs 02            buffer size in dedicated ram: 256*2^val
00030B- 00             1800          .hs 00            always $00
00030C- 00 03 00 00 1820             .hs 00.03.00.00   next parameter block: l,h,b,z
000310- FF             1840          .hs ff            volume level
```

```
1000  REM
              SINE WAVE GENERATOR
            4 CYCLES X 256 BYTES/CYCLE
              1024 BYTES TOTAL

1010 A = 8192
1020 R = (2 * 3.14159) / 360
1030  FOR J = 1 TO 4
1040  FOR I = 0 TO 361 STEP 1.411765
1050 S =  INT ( SIN (R * I) * 128 + 128): IF S = 0 THEN S = 1
1060  POKE A,S: PRINT S,A:A = A + 1
1070  NEXT : NEXT
```

## Warning about 65816 JMP Indirect.........Bob Sander-Cederlof

I discovered it the hard way: JMP (addr) and JML (addr) do not act quite like I expected them to.

In these two instructions the assembled address is a 16-bit value, telling the processor where to look to find the address you want to jump to. In the first form, JMP (addr), the effective jump address will be a 16-bit value, so the next instruction will be executed within the same bank as the JMP instruction. In the second form, JML (addr), the effective jump address will be a 24-bit value, specifying the new bank as well as the position in the bank.

But which bank does the processor look in to find the indirect address? There are three possibilities: it could look in the bank specified by the Data Bank Register, the bank specified by the Program Bank Register, or it could always look in Bank $00.

I ran into this after successfully using the JMP (addr,X) and JSR (addr,X) forms, in which the processor looks for an address table in the bank specified by the Program Bank Register. This is natural, because the table of addresses being accessed most likely is in the same bank as the JSR or JMP. It is only slightly restrictive, in that it requires me to be CERTAIN the table of addresses is in the same bank.

Naturally I assumed the JMP (addr) worked the same way. Naturally my assumption was wrong, and wasted a lot of time. Actually, the processor always looks in bank $00 for the indirect address when you use JMP (addr) or JML (addr). Bank $00? I don't know why. But it is right there in black and white on pages 379 and 383 in the Eyes & Lichty book. Strange.

The only rationale I can think of is that the indirect address might be stored in page zero. Come to think of it, this is a pretty common practice and hence it is probably a pretty good thing the processor works this way. One more thing to remember, though.

Here is a table showing all of the JMP and JSR forms, with the page number from the Eyes & Lichty book, the opcode value, the bank used for the indirect address, and the bank jumped into:

| Eyes | Opcode & Syntax | | indirect bank | destination bank |
|---|---|---|---|---|
| 379 | 4C | JMP addr | N/A | PBR |
| 383 | 6C | JMP (addr) | $00 | PBR |
| 382 | 7C | JMP (addr,X) | PBR | PBR |
| 385 | 5C | JMP long | n/a | stated |
| | or | JML long | | |
| 384 | DC | JML (addr) | $00 | stated |
| | | | | |
| 379 | 20 | JSR addr | n/a | PBR |
| 382 | FC | JSR (addr,X) | PBR | PBR |
| 385 | 22 | JSR long | n/a | stated |

By the way, I used the syntax JML (addr) for the long absolute-indirect Jump. Other assemblers use the syntax JMP [addr]. I chose the bracketless form because the older Apples have no left-bracket on the keyboard.

The form JMP (addr) is used in some existing programs written for the //e and older machines, referencing addresses which are within the body of the program. One such example is inside the code of BASIC.SYSTEM, where it stores the address of the code to process a command in a variable (not in page zero), and then JMPs indirect. Programs which do things like this will not run correctly if they are relocated to higher banks in a blind fashion, by merely setting the PBR and DBR to the new bank and expecting it to run. As I said, I learned the hard way.

You can make it work by setting X=0 and using the JMP (addr,X) form.

I mentioned using the JSR and JMP (addr,X) forms. These are very useful when you have a command processor that needs to branch or call according to a command index. We used to have to use a method which pushed an address from a table on the stack and then did an RTS. For example, a JSR COMMAND.DISPATCHER could call a command processor like this:

```
COMMAND.DISPATCHER
       LDA command.number
       ASL            double it
       TAX
       LDA command.table+1,X   high byte
       PHA
       LDA command.table,X     low byte
       PHA
       RTS     "Jump" to the command routine
command.table
       .DA cmd.0-1,cmd.1-1,cmd.2-1
       .DA cmd.3-1,cmd.4-1,cmd.5-1
```

Now we can do it an easier way. For example, the code segment:

```
       LDA command.number
       ASL            double it
       TAX
       JSR (command.table,X)
```

can call any of a series of subroutines whose addresses are in a list like this:

```
       command.table
              .DA cmd.0,cmd.1,cmd.2
              .DA cmd.3,cmd.4,cmd.5
```

Eight Queens..........................................Phil Goetz

"Eight Queens" is a classic chess puzzle. The Eight Queens problem is, how many ways can you put eight queens on a chessboard so that no queen can capture another? Obviously, each queen will have to be in a different row. Still, there are 8^8 = 16,777,216 ways to put eight queens on a board with each one in a different row. This is clearly a computation-intensive problem.

If you were to try to solve this problem in LISP on an 8-Megabyte VAX system (as AI programmers have a nasty tendency to do), you would end up with a recursive routine which might give you an answer the same day. I was curious how fast an Apple II can solve it, especially since there are 8 positions in each row of a chessboard, and 8 bits in a byte....

I wrote a brute-force routine. My chessboard is a 8*9 array. At all times, exactly one of the 9 positions in each of the 8 rows has a queen. The rightmost (9th) position in each row (held by the carry) is a placeholder which does not count toward a solution. It works like this:

```
             Put all queens in the 9th position.
             X=1
     Queen:  Rotate the queen in row X left.
             If the queen was rotated from the leftmost
                 position back into the 9th, then:

     Backtrack:      X=X-1
                     If X=0 then end.
                     Go to Queen.

             If the queen in row X is in the same column
                 or diagonal as another, go to Queen.
             If X=8:
                    Print this solution.
                    Go to Backtrack.
             X=X+1
             Go to Queen.
```

As written, QUEENS takes 19 seconds to display all 92 solutions, and just 1.9 seconds to find them without displaying. The mathematically minded may say, "But 92 is not a multiple of 8!" Since a chessboard can be flipped and rotated to get 8 symmetries, you expect a multiple of 8 solutions. But one of the solutions is the same when rotated 180 degrees, thus it provides only 4 symmetrical solutions. There are 12 unique solutions in all.

My display routine, in lines 1620 through the end, works best in 40-column mode. It displays what really looks like a chess board, using inverse blank or inverse Q for one color and normal blank or Q for the other color.

Bob S-C wrote an alternate display routine, shown after my listing as lines 1900 and following, which displays the solutions in numeric form. This is considerable faster, and all 92 solutions will fit on one 80-column screen. In Bob's notation, one digit is displayed for each row of the chess board. The digit shows the position of the queen on that row, with 0 being the leftmost square and 7 being the rightmost square. Bob used conditional assembly to control which display routine is assembled. When line 1000 says "GOETZ .EQ 1" my routine is assembled; when it says "GOETZ .EQ 0" Bob's routine is assembled.

```
01-                 1000 GOETZ  .EQ 1
                    SAVE S.EIGHT QUEENS.RBSC
                    1020 *-------------------------------
10-                 1030 BOARD       .EQ $10 thru $17
                    1040 *-------------------------------
32-                 1050 MON.INVFLG .EQ $32
FD8E-               1060 MON.CROUT  .EQ $FD8E
FDED-               1070 MON.COUT   .EQ $FDED
                    1080 *-------------------------------
                    1090 QUEENS
0800- A2 07         1100         LDX #7
0802- A9 00         1110         LDA #0
0804- 95 10         1120 .0      STA BOARD,X
0806- CA            1130         DEX
0807- 10 FB         1140         BPL .0
                    1150 *---Next row down---------------
0809- E8            1160 .1      INX            NEW ROW
080A- E0 08         1170         CPX #8
080C- F0 3E         1180         BEQ .9         ...Finished, have a solution
080E- 38            1190         SEC            INTRODUCE 1 QUEEN INTO ROW
                    1200 *---Next position in row---------
080F- 36 10         1210 .2      ROL BOARD,X
0811- B0 34         1220         BCS .8         THIS ROW FAILED, BACKTRACK
0813- 8A            1230         TXA
0814- F0 F3         1240         BEQ .1         ...First row, any position okay
                    1250 *---Test if any in same column---
0816- A8            1260         TAY            START WITH CURRENT ROW
0817- B9 10 00      1270         LDA BOARD,Y
081A- D9 0F 00      1280 .3      CMP BOARD-1,Y
081D- F0 25         1290         BEQ .7         ...Same column
081F- 88            1300         DEY
0820- D0 F8         1310         BNE .3
                    1320 *---Test if any in same /--------
0822- 8A            1330         TXA            START WITH CURRENT ROW
0823- A8            1340         TAY
0824- B9 10 00      1350         LDA BOARD,Y
0827- 4A            1360 .4      LSR            CHECK "/" DIAGONAL
0828- B0 08         1370         BCS .5         DIAGONAL RAN OFF EDGE
082A- D9 0F 00      1380         CMP BOARD-1,Y
082D- F0 15         1390         BEQ .7         ...ON SAME DIAGONAL
082F- 88            1400         DEY
0830- D0 F5         1410         BNE .4
                    1420 *---Test if any in same \--------
0832- 8A            1430 .5      TXA            START WITH CURRENT ROW
0833- A8            1440         TAY
0834- B9 10 00      1450         LDA BOARD,Y
0837- 0A            1460 .6      ASL            CHECK "\" DIAGONAL
0838- B0 CF         1470         BCS .1         ...Diagonal ran off edge
083A- D9 0F 00      1480         CMP BOARD-1,Y
083D- F0 05         1490         BEQ .7         ...ON SAME DIAGONAL
083F- 88            1500         DEY
0840- D0 F5         1510         BNE .6
                    1520 *---This position looks good-----
0842- F0 C5         1530         BEQ .1         ALWAYS
                    1540 *---This position failed---------
0844- 18            1550 .7      CLC
0845- 90 C8         1560         BCC .2         ...Always
                    1570 *---Backtrack-------------------
0847- 18            1580 .8      CLC
0848- CA            1590         DEX            BACKTRACK
0849- 10 C4         1600         BPL .2         STILL A CHANCE
084B- 60            1610         RTS            NO MORE SOLUTIONS
```

```
                        1620 *---Print the solution-----------
                        1630      .DO GOETZ
084C- 20 8E FD          1640 .9        JSR MON.CROUT
084F- A2 00             1650           LDX #0
0851- B5 10             1660 .10       LDA BOARD,X
0853- 48                1670           PHA
0854- A0 08             1680           LDY #8
0856- A5 32             1690 .11       LDA MON.INVFLG
0858- 49 C0             1700           EOR #$C0
085A- 85 32             1710           STA MON.INVFLG
085C- 68                1720           PLA
085D- 0A                1730           ASL
085E- 48                1740           PHA
085F- A9 A0             1750           LDA #" "
0861- 90 02             1760           BCC .12
0863- A9 D1             1770           LDA #"Q"
0865- 20 ED FD          1780 .12       JSR MON.COUT
0868- 88                1790           DEY
0869- D0 EB             1800           BNE .11
086B- 68                1810           PLA
086C- A5 32             1820           LDA MON.INVFLG
086E- 49 C0             1830           EOR #$C0
0870- 85 32             1840           STA MON.INVFLG
0872- 20 8E FD          1850           JSR MON.CROUT    NEXT ROW
0875- E8                1860           INX
0876- E0 08             1870           CPX #8
0878- 90 D7             1880           BCC .10
                        1890     .ELSE
                        2060     .FIN
087A- 4C 47 08          2070           JMP .8       Will CLC, set X=7, go to .2
                        2080 *--------------------------------
```

```
00-                     1000 GOETZ  .EQ 0

                        1630     .DO GOETZ
                        1890    .ELSE
084C- A2 00             1900 .9        LDX #0
084E- B5 10             1910 .10       LDA BOARD,X
0850- A0 00             1920           LDY #0
0852- 0A                1930 .11       ASL
0853- B0 03             1940           BCS .12
0855- C8                1950           INY
0856- D0 FA             1960           BNE .11
0858- 98                1970 .12       TYA
0859- 09 B0             1980           ORA #"0"
085B- 20 ED FD          1990           JSR MON.COUT
085E- E8                2000           INX
085F- E0 08             2010           CPX #8
0861- 90 EB             2020           BCC .10
0863- A9 A0             2030           LDA #" "
0865- 20 ED FD          2040           JSR MON.COUT
0868- 20 ED FD          2050           JSR MON.COUT
                        2060     .FIN
086B- 4C 47 08          2070           JMP .8       Will CLC, set X=7, go to .2
                        2080 *--------------------------------
```

## Nifty Sonic Signals.......................Bob Sander-Cederlof

Some years ago I wrote version 2 of the PromGramer software for SCRG. One of the new features I added, at the request of Phil's wife, was a pair of termination tunes. If an EPROM successfully programs, the program twiddles Apple's speaker just right to make a cheerful four-note fanfare. Failure, on the other hand, sounds a six-note SS siren.

I call the two tunes NICE and NASTY, and I thought you might like them for your own programs. If not, the same driver can be used to make other tunes. You can play the NICE tune by calling NICE, or NASTY by calling NASTY.

The tunes are encoded in lines 1280-1320. Each note takes two bytes, the first specifying the length of a half-cycle and the second specifying the number of half-cycles. In other words, pitch and duration. I arrived at the numbers by counting machine cycles for the loops, figuring in the Apple clock speed and the frequency for middle-A, and calculating a few notes by hand. Not very sophisticated, but it works nicely.

I decided to try writing a different one, using a technique that plays equal length notes. Lines 1600-1730 are the note player. Lines 1750-1820 call on it to play the NICE tune. PLAY.NOTE.A counts down a duration count simultaneously with the pitch count, so that you get the same total duration regardless of the pitch. This method is not original with me, but I don't remember where it came from. (Maybe the original Apple documentation.)

The $C030 Apple speaker may be obsolete beside the IIgs Ensoniq chip, but I still like it. R2D2 was no Caruso, either....

```
                    1000 *--------------------------------
                    1010 *SAVE S.SIGNAL.TUNES
                    1020 *--------------------------------
00-                 1030 T1      .EQ $00
01-                 1040 T2      .EQ $01
                    1050 *--------------------------------
0800- A0 00         1060 NICE    LDY #TUNE.NICE
0802- 2C            1070         .HS 2C
0803- A0 09         1080 NASTY   LDY #TUNE.NASTY
                    1090 PLAY.TUNE
0805- B9 22 08      1100 .1      LDA TUNES,Y
0808- F0 17         1110         BEQ .4
080A- 85 00         1120         STA T1
080C- B9 23 08      1130         LDA TUNES+1,Y
080F- 85 01         1140         STA T2
                    1150 *---PLAY THE NOTE----------------
0811- AD 30 C0      1160 .2      LDA $C030
0814- A6 00         1170         LDX T1
0816- CA            1180 .3      DEX
0817- D0 FD         1190         BNE .3
0819- C6 01         1200         DEC T2
081B- D0 F4         1210         BNE .2
                    1220 *---NEXT NOTE--------------------
081D- C8            1230         INY
081E- C8            1240         INY
081F- D0 E4         1250         BNE .1         ...ALWAYS
0821- 60            1260 .4      RTS
                    1270 *--------------------------------
```

```
                      TUNES
00-                   TUNE.NICE  .EQ *-TUNES
0822- BB 8A 94
0825- AD BB 8A
0828- 7C CE 00                   .HS BB.8A..94.AD..BB.8A..7C.CE..00
09-                   TUNE.NASTY .EQ *-TUNES
082B- 94 AD BB
082E- 8A 94 AD
0831- BB 8A 94
0834- AD BB 8A
0837- 00                         .HS 94.AD..BB.8A..94.AD..BB.8A..94.AD..BB.8A..00
16-                   TUNE.LO    .EQ *-TUNES
0838- BB 8A 00                   .HS BB.8A..00
19-                   TUNE.HI    .EQ *-TUNES
083B- 94 AD 00                   .HS 94.AD..00
                      *--------------------------------
                      T
083E- AD 00 C0        .1         LDA $C000
0841- 10 FB                      BPL .1
0843- 8D 10 C0                   STA $C010
0846- 49 B0                      EOR #$B0
0848- C9 08                      CMP #8
084A- 90 01                      BCC .2
084C- 60                         RTS
084D- 38              .2         SEC             00000ABC 1
084E- 6A                         ROR             100000AB C
084F- 6A                         ROR             C100000A B
0850- 6A                         ROR             BC100000 A
0851- 48              .3         PHA
0852- A0 16                      LDY #TUNE.LO
0854- 90 02                      BCC .4          ...LOW
0856- A0 19                      LDY #TUNE.HI
0858- 20 05 08        .4         JSR PLAY.TUNE
085B- 68                         PLA
085C- 0A                         ASL             B C1000000, C 1000000, 1 0000000
085D- D0 F2                      BNE .3          DO B OR C
085F- 4C 3E 08                   JMP .1          FINISHED
                      *--------------------------------
                      PLAY.NOTE.A
0862- AA                         TAX
0863- A9 2E                      LDA /12000
0865- 38                         SEC
0866- 86 00                      STX T1
0868- CA              .1         DEX
0869- D0 05                      BNE .3
086B- A6 00           .2         LDX T1
086D- 2C 30 C0                   BIT $C030
0870- 88              .3         DEY
0871- D0 F5                      BNE .1
0873- E9 01                      SBC #1
0875- B0 F1                      BCS .1
0877- 60                         RTS
                      *--------------------------------
0878- A9 5D           M.NICE     LDA #$5D
087A- 20 62 08                   JSR PLAY.NOTE.A
087D- A9 4A                      LDA #$4A
087F- 20 62 08                   JSR PLAY.NOTE.A
0882- A9 5D                      LDA #$5D
0884- 20 62 08                   JSR PLAY.NOTE.A
0887- A9 3E                      LDA #$3E
0889- 4C 62 08                   JMP PLAY.NOTE.A
                      *--------------------------------
                      M
088C- A9 14                      LDA #20
088E- 48              .1         PHA
088F- 2C 00 C0                   BIT $C000
0892- 30 0A                      BMI .2
0894- 20 62 08                   JSR PLAY.NOTE.A
0897- 68                         PLA
0898- 18                         CLC
0899- 69 01                      ADC #1
089B- D0 F1                      BNE .1
089D- 60                         RTS
089E- 8D 10 C0        .2         STA $C010
08A1- 68                         PLA
08A2- 60                         RTS
                      *--------------------------------
```

## A New BASIC for the IIgs

Apple has announced their new BASIC interpreter for the IIgs. No, it is not an upgraded Applesoft. It is a totally different version, appearing to me to be based on the old Apple /// Business Basic. GSBASIC runs under ProDOS-16, and supports all of the IIgs tools and features.

Features include 9- or 15-digit precision in floating point with exponent range from -308 to +308, plus short and long integers (sounds like SANE); PRINT USING; variable names up to 30 characters long; use of labels rather than line numbers for GOTO and GOSUB; editing features including search/replace; built-in disk file I/O commands; direct statements for access to the various IIgs Tools; and, you guessed it, more! The scheme for linking to assembly language code is the same as in Apple /// Business Basic, via INVOKE, EXFN, and PERFORM statements.

Negatives include the new tradition of extremely long booting time; total in-compatibility with Applesoft programs; requirement for a minimum (MINIMUM!) of 512K RAM; will not work under DOS, ProDOS, or any machine other than a IIgs. It turns out trying to use GSBASIC with less than 1024K is not really reasonable.

The pre-release disk includes ProDOS-16 version 1.3 with the FINDER, GSBASIC, and a sample program which displays four Super Hi-Res pictures under mouse control.

Some interesting trivia: instead of CATALOG and CAT, we now have CATALOG and DIR. Both display an 80-column list of files in the current directory. CATALOG looks a lot like our old ProDOS-8 favorite, and DIR is just a slight revision of it. DIR displays a tiny icon in front of each filename, leaves out the creation date, rounds up the file size to the nearest Kbytes, and spells out the access bits.

If you hit Ctrl-RESET, you are in for a surprise. You get the marvelous message "CANNOT RESET -- $0602", a big beep, and the see-sawing Apple. If you reset again you end up in the monitor. No problem, just boot again, wait five minutes for it all to reload, and then type in your whole program over again!

GSBASIC is available now in pre-release form (beta version 1.0B4) from APDA (Apple Programmer's Development Association), for $50. This is a very low price for such a large system. Applesoft originally cost $100, but of course it came on a ROM card. Applesoft is only 12K bytes long. GSBASIC is currently about 59K bytes long, and growing.

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are $1.80 each (other countries inquire for postage). A subscription to the newsletter and the Monthly Disk containing all source code is $64 per year in the US, Canada and Mexico, and $87 to other countries.